Theoretical Foundations of Immune-Inspired Randomized Search Heuristics for Optimization

Zarges, Christine

Published in:
Theory of Evolutionary Computation

Publication date:
2019

Citation for published version (APA):
Zarges, C. (2019). Theoretical Foundations of Immune-Inspired Randomized Search Heuristics for Optimization. In B. Doerr, & F. Neumann (Eds.), *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization* (pp. 443-474). Springer Nature.

Theoretical Foundations of Immune-Inspired Randomized Search Heuristics for Optimization*

Christine Zarges
Department of Computer Science
Aberystwyth University
Aberystwyth SY23 3DB, UK

c.zarges@aber.ac.uk

Abstract

Artificial immune systems are a class of nature-inspired algorithms based on the immune system of vertebrates. They have been used in a large number of different areas of application, most prominently learning, classification, pattern recognition, and (function) optimization. In the context of optimization, clonal selection algorithms are the most popular and constitute an interesting and promising alternative to evolutionary algorithms. While structurally similar, they offer very different features and capabilities. Over the last decade, significant progress has been made in the theoretical foundations of clonal selection algorithms. This chapter gives an overview of the state of the art in the theory of artificial immune systems with a focus on optimization. It provides pointers to corresponding articles where more details and proofs can be found.

1 Introduction

Artificial immune systems (AIS) are derived from various immunological theories, namely the clonal selection principle [4], immune network theory [46], and the danger theory [22]. Besides the natural tasks of anomaly detection and classification, they are often applied to function optimization. In the latter context, most immune-inspired randomized search heuristics are based on the clonal selection principle [4], a theory which describes the basic features of an adaptive immune response to invading pathogens (antigens). The most popular clonal selection algorithms to tackle optimization problems include CLONALG [23], Opt-IA [18], the B-cell algorithm [47], and MISA [7]. All these algorithms are population-based. The input is usually represented by a population of antigens; a population of immune cells represents candidate solutions of the problem considered. Various aspects of clonal selection are used in these immune-inspired algorithms, for example mutations of different types of immune cells found in the immune system, resulting in a large number of very different approaches that share a common biological inspiration. Many of these algorithms resemble evolutionary algorithms from a structural point of view. However, their concrete implementations are usually very different.

This chapter provides an overview of the state of the art in the theory of immune-inspired randomized search heuristics in discrete search spaces. Most theoretical studies

*This is a preliminary version of a chapter in the upcoming book "Theory of Evolutionary Computation: Recent Developments in Discrete Optimization", edited by Benjamin Doerr and Frank Neumann, to be published by Springer.

so far have concentrated on pseudo-Boolean optimization and classical example functions; however, some initial work on combinatorial optimization (vertex cover, longest common subsequence) exists. We discuss problem definitions, analytical frameworks, and common algorithms and operators in the corresponding sections. However, note that we consider bit strings of length n as the representation, $x \in \{0,1\}^n$, and that $x[i]$ denotes the i-th bit in x (with $i \in \{0,1,\ldots,n-1\}$, denoting the leftmost position in x by $x[0]$ and the rightmost position by $x[n-1]$).

The main part of this chapter will concentrate on performance analyses such as runtime analysis (where we are interested in the number of function evaluations required to locate an optimal solution, called the optimization time) and fixed-budget analysis (where we analyze the expected solution quality for a given budget of function evaluations). We therefore start with a brief overview of other related publications: early theoretical work was particularly concerned with Markov chain models of clonal selection algorithms and convergence analyses, i. e., the study of whether a given algorithm is guaranteed to converge to a global optimum for time $t \to \infty$. Based on Markov chain theory, Villalobos-Arias et al. [56, 57] proved convergence of the multi-objective clonal selection algorithm MISA [7] under the condition that the algorithm maintains an elitist memory throughout the search process. Later, using a similar approach, convergence results for the B-cell algorithm [47] based on a Markov model for contiguous hypermutations were presented [5, 6]. Cutello et al. [17] considered a more general framework of immune algorithms and examined conditions sufficient for their convergence. They provided some problem-independent upper bounds for their class of immune algorithms, but pointed out that such analyses should be related to some problem class and its characteristics in order to give useful insights. More recently, Hong and Kamruzzaman [29] used martingale theory to prove convergence for a class of elitist clonal selection algorithms. Timmis et al. [55] provided a survey of early theoretical advances and pointed out that runtime analysis would be much more useful than convergence analysis. The remainder of this chapter will therefore concentrate on these more advanced results.

The vast majority of work to date has concentrated on AIS for optimization. In this context, two defining aspects of AIS have been particularly considered: hypermutation operators (Section 2) and a diversity mechanism called aging (Section 3). However, more recently, insights into the interplay between different operators have allowed the first analyses of "complete" AIS as published in the literature (Section 4).

For the sake of completeness, we remark that theoretical studies in other subareas of artificial immune systems exist. In the context of classification, we refer the reader to the above-mentioned survey by Timmis et al. [55] and more recent work by Elberfeld and Textor [24, 54] and Gu et al. [27]. Moreover, a relatively large number of surveys provide a comprehensive view of the various application areas of artificial immune systems. They include general overviews and introductions to the field [22, 19, 20, 53], as well as more specialized surveys of, for example, optimization [1] and security [25].

2 Theoretical Analyses of Hypermutations

Mutation in AIS is very different from mutation in other randomized search heuristics, for example evolutionary algorithms. While in evolutionary algorithms generally moderate mutation probabilities are employed, AIS incorporates mutations at a high rate, so-called hypermutations. Different types of immune-inspired mutation operators with roots in different (processes of the) immune systems can be found in the literature. Among the most prominent classes are inversely fitness-proportional mutations (Section 2.1), contiguous hypermutations (Section 2.2), and hypermutations with mutation potential (Section 2.3). We will discuss different variants of these operators in the following.

All the results presented in this section consider immune-inspired mutation operators in minimalistic algorithmic frameworks to study them in as much isolation as possible.

These frameworks include in particular the (1+1) framework shown in Algorithm 1 and a simple population-based (μ+1) framework as described in Algorithm 2. The performance of hypermutation operators has particularly been compared with local search (flipping exactly one random bit) and standard bit mutations (flipping each bit with probability $1/n$).

Algorithm 1: (1+1) framework

```
Choose x ∈ {0,1}^n uniformly at random.
repeat
    Create offspring y := mutate(x).
    if f(y) ≥ f(x) then
        Set x := y.
until some termination criterion is met
```

Algorithm 2: (μ+1) framework

```
Parameters : Population size μ
Choose x_1, ..., x_μ ∈ {0,1}^n independently, uniformly at random.
Let P := {x_1, ..., x_μ}.
repeat
    Choose x ∈ P uniformly at random.
    Create offspring y := mutate(x).
    Choose z ∈ P with minimum fitness.
    if f(y) ≥ f(z) then
        Let P = P\{z} ∪ {y}.
until some termination criterion is met
```

2.1 Inversely Fitness-Proportional Mutations

The idea of inversely fitness-proportional mutations derives directly from the widely accepted clonal selection principle [4]. It aims to balance exploration and exploitation of the search space by using an individual mutation rate for each search point (immune cell) in the population, i. e., focusing on exploitation for good search points that are hopefully close to a local or global optimum and focusing on exploration otherwise. As a result, search points in "better" regions of the search space are only subject to small mutations, while for search points located far away from optimal regions larger mutation rates are used.

Inversely fitness-proportional mutation operators exist in both continuous and discrete versions, but to the best of our knowledge theoretical studies so far have concentrated on discrete settings or, more precisely, pseudo-Boolean optimization. Here, the mutation rate is a function that depends on the (normalized) fitness of a search point and determines the probability for each bit in a given bit string to be mutated (see Algorithm 3). The relationship between mutation rate and fitness is not required to be inversely proportional in a strict mathematical sense, but only needs to follow the rule that the higher the fitness the smaller the mutation rate, and vice versa.

Algorithm 3: Inversely fitness-proportional mutations

```
Input: Search point x ∈ {0,1}^n; fitness-dependent mutation rate p(v)
Output: Mutated search point x ∈ {0,1}^n
Let v := normalize(f(x)) ∈ [0,1].
Let y := x.
for i := 0 to n − 1 do
    With probability p(v) set y[i] := 1 − y[i].
```

Inversely fitness-proportional mutations are a key ingredient of CLONALG [23]. For the case of maximization, the operator uses the inverse of an exponential function to establish a relationship between the mutation probability and the normalized fitness value v:

$$p_{\text{CLONALG}}(v) = \exp(-\rho \cdot v), \tag{1}$$

where ρ is a so-called decay parameter that controls the smoothness of the exponential function and needs to be set to a value appropriate for the problem considered. A similar operator with a slightly different parameterization is used in Opt-AiNet [21], an immune-inspired algorithm based on immune network theory [46]:

$$p_{\text{Opt-AiNet}}(v) = \exp(-v)/\rho. \tag{2}$$

Here, the parameter ρ is not incorporated into the exponent of the exponential function but rather used to scale its result, leading to a very different impact of ρ on the optimization process [61].

Zarges [59, 61] examined the role of the decay parameter ρ for these two operators on the OneMax problem in a simple $(1 + 1)$ framework (see Algorithm 1). Constant decay values ρ as well as values logarithmic and linear in the length of the bit string were considered. Using, among others, drift arguments, it was shown that both operators are very sensitive to parameterization and, if parameterized inappropriately, very bad at hill-climbing.

For maximization problems, the standard normalization method [23] is to divide the fitness by the best fitness in the current population or by the best fitness seen so far. For a single search point, Zarges [59, 61] used the optimal value of the fitness function considered instead and argued that using an upper bound on the fitness would be appropriate if the optimal value was not known. However, it was noted that using an upper bound leads to generally larger mutation probabilities, while the use of the best current fitness reduces them.

For CLONALG mutations, it was proven that, with overwhelming probability, the algorithm using constant and linear settings of ρ is unable to locate the optimum of OneMax within a polynomial number of iterations. For a constant ρ, this is because the mutation probabilities grow much too large to be effective, while a linear ρ leads to exponentially small mutation rates, rendering the algorithm unable to perform any search at all. In addition to these two extreme cases, $\rho = \ln n$ was considered. It was noted that this parameterization yields reasonable values for the mutation rate between $1/2$ and $1/n$. While one can still observe a large negative drift for this setting (and consequently the algorithm is unable to optimize OneMax in polynomial time with high probability), the algorithm demonstrates much better performance in practice, as the probability of not finding the optimum within a polynomial number of iterations converges much more slowly to 1 than it does for constant ρ. In fact, in experiments the performance was comparable to that of standard bit mutations up to bit string lengths of about 10^5.

For Opt-AiNet mutations, a constant ρ results in roughly the same mutation rates as already seen for the case of CLONALG, and thus the algorithm is also not efficient on OneMax with this setting. However, using $\rho = \Theta(n)$ yields mutation rates of $\Theta(1/n)$ for all possible fitness values. Thus, we get an optimization time of $\Theta(n \log n)$ for this case, which can easily be shown by using fitness-level arguments and adapting previous analyses for standard bit mutations.

Later, the CLONALG hypermutation operator was analyzed in a $(\mu + 1)$ framework (see Algorithm 2), using the current best fitness value for normalization [60]. Setting $\rho = \ln n$ (as this leads to reasonable mutation rates for OneMax [59]) and using fitness-level arguments, it was shown that even a population size of 2 considerably improves the performance on the OneMax problem and, more generally, a class of smooth integer functions of unitation, i. e., a class of functions where the function value depends only on the number of 1-bits in the bit string and neighboring points in the search space have

similar function values. A key insight for this result is that for this parameterization the behavior of the current best search point in the population mimics that of standard bit mutations.

A matching lower bound was proven by bounding the bandwidth of the fitness values in the current population using inductive arguments and by employing the technique of analyzing randomized family trees. Here, an important insight is that the rate of inversely fitness-proportional mutations depends directly on the structure of the population or, more precisely, the difference between the best and the worst fitness value. If this difference is small enough, the expected number of flipping bits during a single iteration can be bounded by a constant.

While fitness-dependent mutations are common in immune-inspired randomized search heuristics, they have only recently emerged in the context of evolutionary computation. For example, Oliveto et al. [51] analyzed a rank-based mutation operator, an alternative to using normalized function values to reduce the effect of large differences in absolute function values. Böttcher et al. [2] derived an optimal adaptive mutation rate for the LeadingOnes problem that has the form of an inversely fitness-proportional mutation rate, $p_{\text{simple}}(v) = 1/(v+1)$ with $v = f(x)$. The same mutation rate is efficient for the OneMax problem [61]. However, a Hamming-distance-based mutation rate maximizing the probability of finding the global optimum of OneMax in a single mutation step, i. e., $p_{\text{Hamming}}(v) = v/n$ with $v = n - f(x)$, yields exponential optimization time.

Jansen and Zarges [43] considered inversely fitness-proportional mutations in the context of fixed-budget analysis, where, instead of the expected time needed for optimization, the expected performance within a given time frame is analyzed. They showed that CLONALG mutations outperform local search at the beginning of the run, but are eventually overtaken later in the run. This insight was used to devise a hybrid algorithm that starts with CLONALG mutations and switches to local search when progress stagnates. These results are discussed in more detail in Section 5.5 of this book.

2.2 Contiguous Hypermutations

Contiguous hypermutations were introduced as part of the B-cell algorithm [47]. They were inspired by the observation that mutation of B-cell receptors (a type of immune cell) often focuses on specific regions of the receptor. To mimic this behavior, the mutation operator first selects a contiguous region of the search point's representation and restricts the mutation to this region. The use of contiguous hypermutations is limited to discrete search spaces and employs a bit string representation. Mutation flips all bits within the chosen region with a given probability $r \in [0, 1]$ and does not change any bit that is outside of this region. It has been noted that, depending on its parameterization, this mutation operator can easily be trapped in local optima [6, 37]: if $r = 1$, there might not exist a mutation leading from a local to a global optimum. However, the following analyses consider this extreme case only.

Jansen and Zarges [34, 37] considered three different variants of this hypermutation operator which differ in the way the contiguous mutation region is determined. The original operator chooses a random starting position p and a random length l of an interval to be mutated (see Algorithm 4). It does not wrap around, and thus has a strong positional bias and strongly different mutation probabilities for mutations of single bits depending on their location: bits towards the end of the bit string have a higher probability of being mutated during a mutation. As such a bias is considered undesirable unless it suits known problem characteristics, this observation motivates the definition of a variant that wraps around and thus has no positional bias at all (see Algorithm 5). A third variant selects random start and end points for the mutation region (see Algorithm 6). Similarly to the original operator, this variant has a strong positional bias – here, bits towards the middle of the bit string have a higher probability of being mutated. All three variants

have in common that, in expectation, they flip a linear number of bits. The probability of performing a single bit mutation is $\Theta(1/n^2)$ in all three cases (with the exception of the version that does not wrap around, where the probability of flipping only the last bit is $\Theta(1/n)$). This is considerably smaller than the corresponding probability for standard bit mutations and explains why contiguous hypermutations are also bad at simple hill-climbing.

Algorithm 4: Contiguous hypermutations, not wrapping around [5]

Input: Search point $x \in \{0,1\}^n$; mutation probability $r \in (0,1]$
Output: Mutated search point $x \in \{0,1\}^n$

Select $p \in \{0,1,\ldots,n-1\}$ uniformly at random.
Select $l \in \{0,1,\ldots,n\}$ uniformly at random.
for $k := 0$ *to* $\min\{l-1, n-1-p\}$ **do**
    With probability r, invert the bit $x[p+k]$.

Algorithm 5: Contiguous hypermutations, wrapping around [34]

Input: Search point $x \in \{0,1\}^n$; mutation probability $r \in (0,1]$
Output: Mutated search point $x \in \{0,1\}^n$

Select $p \in \{0,1,\ldots,n-1\}$ uniformly at random.
Select $l \in \{0,1,\ldots,n\}$ uniformly at random.
for $i := 0$ **to** $l-1$ **do**
    With probability r set $x\left[(p+i) \mod n\right] := 1 - x\left[(p+i) \mod n\right]$.

Algorithm 6: Contiguous hypermutations, two hotspots [34]

Input: Search point $x \in \{0,1\}^n$; mutation probability $r \in (0,1]$
Output: Mutated search point $x \in \{0,1\}^n$

Select $p_1 \in \{0,1,\ldots,n-1\}$ uniformly at random.
Select $p_2 \in \{0,1,\ldots,n-1\}$ uniformly at random.
for $k := \min\{p_1,p_2\}$ *to* $\max\{p_1,p_2\}$ **do**
    With probability r, invert the bit $x[k]$.

However, they can have advantages when large mutations are needed: standard bit mutations perform specific b-bit mutations with probability $\Theta(1/n^b)$, while all three variants of contiguous hypermutation achieve this with probability $O(1/n^2)$. Jansen and Zarges [34, 37] investigated this in a rigorous way by presenting different examples for functions where contiguous hypermutations are superior or inferior to the standard bit mutations typically used in evolutionary algorithms in a simple (1+1) framework (see Algorithm 1). Using fitness level and drift arguments, they showed that contiguous hypermutations can drastically outperform standard bit mutations for a previously know family of example functions that require mutations of many bits simultaneously,

$$\mathrm{CLOB}_{b,k}(x) = n \cdot \left(\sum_{h=1}^{k} \sum_{i=1}^{n/(bk)} \prod_{j=0}^{i \cdot b - 1} x\left[(h-1) \cdot \frac{n}{k} + j \right] \right) - \textsc{OneMax}(x),$$

for $b, k, n \in \mathbb{N}$ with $n/k \in \mathbb{N}$, $n/(bk) \in \mathbb{N}$, and $x \in \{0,1\}^n$. Here, contiguous hypermutations yield an optimization time of $O(n^2 \log n)$, while standard bit mutations require time $\Theta(n^b(l/b + \ln k))$. In addition, it was shown that contiguous hypermutations do not necessarily lose a factor of $\Theta(n)$ on functions where mutations of single bits are responsible for optimization: while this is the case for OneMax, contiguous hypermutations lose at most a factor of $\log n$ on LeadingOnes.

Jansen and Zarges [34, 37] investigated the role of initialization for contiguous hypermutations and demonstrated that advantageous starting points with large blocks of

contiguous 0s (e. g., the all-zero bit string 0^n) can speed up optimization for this operator while having nearly no impact on standard bit mutations. However, whether this advantage, which is big in the beginning where it is easy to make progress and decreases towards the end where making progress is much harder, is sufficient to yield an asymptotically smaller expected optimization time depends on how long this advantage can be preserved during the optimization process. It is important to note that all positive results rely on the extreme choice $r = 1$. Thus, it can be concluded that contiguous hypermutations can only play out their strength if r is set to some value at least close to 1.

Jansen and Zarges [43] later also considered contiguous hypermutations in the context of fixed-budget analysis (see Section 5.5 of this book). Revisiting negative results for immune-inspired hypermutations for the simple OneMax function and the observation that careful initialization can speed up the optimization process [37], they demonstrated that contiguous hypermutations can be much more efficient in the beginning of a run when progress is still easy to achieve and, thus, given a limited budget of function evaluations, such mutations can by far outperform a random local search operator. This is mainly due to the fact that hypermutations are able to accumulate many small steps into a single large one, while random local search needs to perform those steps one after each other and thus needs time to catch up. This insight helps to explain the success of seemingly inefficient mutation operators, as in practice the length of a run is usually limited.

The above theoretical results were put into practice by designing a more efficient hybrid search heuristic that applies contiguous hypermutations in the beginning when they can be expected to be more beneficial, and switches to local search when contiguous hypermutations start to become slow. Two strategies to switch the mutation operator were investigated, one directly based on the theoretical findings, the other using the expected progress of the two operators adaptively in each iteration. Experiments showed that both strategies yield noticeable improvements over simple local search if careful initialization is performed, but that the more sophisticated adaptive strategy does not yield any significant advantage. These results are discussed in more detail in Section 5.5 of this book.

Other work on the analysis of contiguous hypermutations includes a runtime and fixed-budget analysis for the highly multimodal HIFF (hierarchical if and only if) problem by Jansen and Zarges [44] and later by Xia and Zhou [58]. Jansen and Zarges [44] showed that under certain conditions contiguous hypermutations can be successful hill-climbers. Using fitness-level arguments, they showed that contiguous hypermutations in a simple (1+1) framework (see Algorithm 1) solve HIFF in time $O(n^3 \log n)$, while random local search does not find an optimum with overwhelming probability. Moreover, they demonstrated that contiguous hypermutations are not outperformed by random local search on HIFF for any given budget by performing a fixed-budget analysis (see Section 5.5 of this book): for small budgets both algorithms have roughly equal performance, but contiguous hypermutations outperform random local search for moderately large budgets. This result is somewhat counterintuitive as at the beginning, i. e., before reaching a local optimum, HIFF can be optimized by simple hill-climbing – something contiguous hypermutations are particularly bad at.

Xia and Zhou [58] additionally considered contiguous hypermutations on Trap functions and the max-cut and minimum s-t-cut problems, again using the simple (1+1) framework in Algorithm 1. They showed that Trap can be optimized in time $O(n^2 \log n)$ and considered a family of graphs for max-cut that can be efficiently optimized using contiguous hypermutations but not using standard bit mutations and a problem-specific local search operator. A similar result was shown for a family of graphs for the minimum s-t-cut problem.

Very recently, Corus et al. [8, 9] derived an easiest function for contiguous hypermutations. Again using the insight that contiguous hypermutations can have advantages

on functions that require mutations of many bits simultaneously, they introduced the following fitness function.

Definition 2.1. Let $L_0 \dot\cup L_1 \dot\cup L_2 \dot\cup \cdots \dot\cup L_l = \{0,1\}^n$ be a partition and let

$$\textsc{MinBlocks}(x) = l - i \text{ for } x \in L_i,$$

with $l = \lfloor n/2 \rfloor + 1$, $L_0 = \{1^n\}$, $L_1 = \{0^n\}$, and $L_i = \{x \in \{0,1\}^n \mid x \text{ contains } i-1 \text{ different 1-blocks}\}$ for each $i \in \{2, 3, \ldots, l\}$.

MinBlocks has a unique global optimum, 1^n, with fitness $\lfloor n/2 \rfloor + 1$. The second best bit string is 0^n, with fitness $l - 1$. Corus et al. [8, 9] presented both runtime and fixed-budget analyses of contiguous hypermutations on this function and showed that MinBlocks is indeed an easiest function using a method introduced by He et al. [28]. The runtime of contiguous hypermutations embedded in a (1+1) framework (see Algorithm 1) on MinBlocks is $\Theta(n^2)$.

MinBlocks turns out to be an asymptotically hardest function for standard bit mutations. Owing to the symmetry of 0- and 1-bits, the $(1 + 1)$ EA (Algorithm 1 using standard bit mutations) will reach 0^n (instead of 1^n) with probability 1/2. In this situation, it needs to flip all bits in a single mutation, resulting in an expected optimization time of at least $n^n/2$. This is only smaller by a factor of at most 2 than the expected optimization time of the $(1 + 1)$ EA on its hardest function, Trap [28].

Finally, Corus et al. [8, 9] discussed a number of hybridizations of standard bit mutations and contiguous hypermutations. These allow one to combine the advantages of the two operators, and yield optimal asymptotic performance on both OneMax and MinBlocks.

Some analyses of contiguous hypermutations consider the mutation operator within the complete B-cell algorithm rather than a minimalistic framework. We discuss these results later in Section 4.1.

2.3 Hypermutations with Mutation Potential

Hypermutations with mutation potential were introduced as a mutation operator in Opt-IA [18]. The main idea behind this kind of mutation is to determine the number of local mutation steps by a given function, the so-called mutation potential. Mutation potentials exist in different flavors, for example static, fitness-proportional, and inversely fitness-proportional. Moreover, they can be restricted to certain regions of the bit string (hypermacromutation).

Algorithm 7 provides pseudocode of four different variants of this mutation operator, defined for minimization problems. For a number M of local mutation steps, the operator sequentially draws M not necessarily distinct positions in the bit string and flips them independently. We distinguish a TABU variant (where the operator is prevented from choosing a specific position two or more times) and a non-TABU variant (where bits can be flipped back in a later mutation step). Moreover, a mechanism often used in conjunction with mutation potentials is the so-called "stop at first constructive mutation" (FCM). Here, a fitness evaluation is performed after every single mutation step and the mutation stops if an improvement has been found (a so-called constructive mutation). It has been shown that the question of whether local mutation steps may undo each other is far less important than the use of the FCM mechanism [10, 15, 40]. We provide more detail in the following.

Algorithm 7: Mutation with mutation potential M (minimization)

Input: Search point $x \in \{0,1\}^n$; flags TABU and FCM
Output: Mutated search point $x \in \{0,1\}^n$

```
Set y = x.
repeat
    if TABU = 0 then
        Select i ∈ {1, ..., n} uniformly at random (u.a.r.).
    else
        Select i ∈ {1, ..., n} u.a.r., i not previously chosen.
    Invert the bit y[i].
    if (FCM = 1) AND (f(y) < f(x)) then
        BREAK
until M times
```

2.3.1 Inversely Fitness-Proportional Mutation Potentials

The first analysis of hypermutations with mutation potential was presented by Jansen and Zarges [40]. They considered all four variants of this operator for an inversely fitness-proportional mutation potential $M_c(v)$, where $c \in]0,1[$ and f_{OPT} is the minimum function value for the fitness function considered, $f\colon S \to \mathbb{R}^+$:

$$M_c(v) = \lceil (1 - f_{\mathrm{OPT}}/v) \cdot c \cdot n \rceil. \tag{3}$$

Using the simple (1+1) framework (see Algorithm 1), they showed that the FCM mechanism is crucial for the performance of mutation potentials even on very simple optimization problems such as ZeroMin, the minimization variant of OneMax. The main reason for this is that hypermutations with mutation potential that do not make use of FCM basically perform a random walk of length equal to the mutation potential – such a random walk has hardly any chance of locating a specific search point (which can easily be proven by using results for the gambler's ruin problem). While adding FCM considerably improves the optimization time, it loses a factor of n in comparison with standard bit mutations: the expected optimization time of Algorithm 1 using hypermutations with mutation potential on ZeroMin is $2^{\Omega(n)}$ without FCM and $\Theta(n^2 \log n)$ with FCM. The upper bounds for the algorithms with FCM were derived using fitness-layer arguments, while the lower bound was obtained by Chernoff bounds and arguments from the classical ballot theorem for the non-TABU variant, and a careful analysis of the underlying random walk for the TABU version.

Moreover, Jansen and Zarges [40] analyzed the ability of the original hypermutation operator with FCM to locate optima precisely and at a large distance from other promising regions of the search space by considering a previously introduced example function called Sp-Target (short path with target). The main idea of this function is that the vast majority of the search space guides the search heuristics towards a path with increasing function values starting from the all-zero bit string 0^n. The global optimum is a large area with a minimum Hamming distance to the path.

A typical run of a search heuristic finds and climbs the path before finding the optimal region. Thus, in order to be able to optimize this function, an algorithm requires the ability to "jump" from the path into the global optimum. It is known that standard bit mutations are unable to efficiently locate the global optimum if the distance is $\omega(\log n / \log\log n)$. Jansen and Zarges [40] proved that hypermutations with mutation potential yield an optimization time of $O(n^3)$ provided that c in $M_c(v)$ is chosen large enough.

More recently, Corus et al. [14] compared different variants of inversely fitness-proportional mutation potentials based on Hamming distance and fitness difference. They showed that

a potential that increases exponentially with the Hamming distance to the optimum (called M_{expoHD}) is most promising and argued that using Hamming distance instead of fitness difference also comes with the advantage of robustness to scaling of the fitness function. In comparison with static mutation potentials, they showed a considerable speedup for all inversely fitness-proportional variants on standard unimodal example functions, for which the global optimum is known. In addition, M_{expoHD} was considered in situations, where the global optimum is unknown. Using the best found solution to estimate the mutation rate and combining M_{expoHD} with hybrid aging (see Section 3.2), it was demonstrated that the algorithm might not be able to identify new local optima on slopes that lead away from previous ones. As a consequence, a symmetric version of M_{expoHD} was introduced and shown to be effective on two well-known bimodal example functions, $\textsc{Cliff}_d$ and TwoMax.

2.3.2 Static Mutation Potentials

Corus et al. [10, 15] presented a first detailed study of static mutation potentials. They proved for a static mutation potential (where the number of bits flipped is linear in the problem size, $c \cdot n$ for constant $c > 0$) and the TABU variant that unless the FCM mechanism is applied, hypermutations with mutation potential require exponential expected time to optimize any function with a polynomial number of optima. They argued that the search point created by such hypermutations is uniformly distributed over all search points with distance $c \cdot n$ to the search point it was derived from. Since there are exponentially many such search points, the probability of a specific outcome is exponentially small. Corus et al. [10, 15] pointed out that this result could easily be extended to other types of mutation potential such as inversely fitness-proportional [40] and fitness-proportional mutation potentials [16]. In [15], the authors additionally suggested that it may be beneficial to call a mutation "constructive" if the fitness is at least as good (instead of strictly better) to improve the algorithm's exploration capabilities.

Moreover, Corus et al. [10, 15] showed that the expected optimization time if FCM is used is at most larger by a linear factor than the upper bound obtained for random local search (with any neighborhood size) via fitness-level arguments. This demonstrates that it is sufficient to analyze random local search instead of hypermutations with such a mutation potential (which is often easier) to achieve a valid upper bound on the expected optimization time. Corus et al. [10, 15] showed that these bounds are tight for easy example functions such as OneMax and LeadingOnes.

Finally, Corus et al. [10, 15] compared hypermutations with mutation potential with standard mutation- and crossover-based evolutionary algorithms on the $\textsc{Jump}_k$ and $\textsc{Cliff}_d$ functions. They proved that their hypermutations operator could exponentially speed up the process of escaping from local optima, particularly in cases where the jump is hard to perform. However, the upper bound on the expected optimization time was still exponential in the distance between the local and the global optimum.

Very recently, Corus et al. [11, 13] presented an analysis for the NP-hard number partitioning problem. They showed that, due to its ability to escape from local optima, a simple artificial immune system using static hypermutations with mutation potential is able to efficiently solve a class of problem instances that are known to be hard for random local search and evolutionary algorithms using standard bit mutations. More importantly, they proved that such an artificial immune system is a randomised polynomial time approximation scheme (for $\varepsilon = \omega(n^{-1/2})$), i. e., it guarantees an approximation ratio of $(1 + \varepsilon)$ for any problem instance in expected time polynomial in the problem size and exponential in $1/\varepsilon$. The authors pointed out that, to the best of their knowledge, this was the first time performance guarantees were proven for an artificial immune system on a classical combinatorial optimization problem.

2.3.3 Variants of Mutation Potentials

Based on some of the above findings, novel variants of hypermutations with mutation potential were introduced. Jansen and Zarges [40] proposed an improved version of the mutation potential based on ranks that allows one to parameterize the trade-off between efficiency in local search and the ability to perform huge changes in a single mutation:

$$\hat{M}_{c,\rho}(v_i) = \lceil (1 - n^\rho/(n^\rho + i - 1)) \cdot c \cdot n \rceil, \tag{4}$$

where i is the rank of the fitness value considered among all fitness values in the search space, and ρ controls the degree of mutation aversion the hypermutation operator has. It was proven that for $\rho > 1$ the expected optimization time on ZeroMin with FCM decreases to $\Theta(n \log n + n^{3-\rho})$.

More recently, Corus et al. [12] proposed a "fast" variant of hypermutations with mutation potential, where instead of deterministically performing a fitness evaluation after each bitflip, the fitness after the i-th bitflip is only evaluated with probability roughly $p_i \approx \gamma/i$. In doing so, fewer function evaluations are "wasted" during the hypermutation process, particularly for easy problems, for which local search strategies are efficient. The effectiveness of two variants of this operator coupled with and without FCM was demonstrated by analyzing problems that had been considered previously for hypermutations with mutation potential and Opt-IA (see Section 4.2) and recommendations for setting the parameter γ were provided. Moreover, it was demonstrated how upper bounds for "fast" hypermutations with mutation potential could be derived from upper bounds for random local search that were obtained via fitness-level arguments.

3 Theoretical Analyses of Aging Operators

Aging operators require that each search point in the population is equipped with an individual age that is increased by 1 in each iteration of the search heuristic. A maximum lifespan τ is introduced, and each search point with an age exceeding τ is removed from the current population, making room for new and perhaps more promising search points. The mechanism of aging is thought of as increasing the diversity of the population and it is hoped that it will be helpful for multimodal problems where simpler search heuristics may get stuck in local optima.

Different variants exist, and static pure aging and stochastic aging have both been used within Opt-IA [18]. Both strategies usually have in common that the initial age of a new search point is set to 0 only if its function value is strictly larger than the function value of the search point it was derived from (the parent); otherwise, it inherits the age of this search point. This scheme is intended to give an equal opportunity to each improving new search point to explore the landscape effectively. Alternatively, each new search point can be assigned age 0; however, this is more common in evolutionary computation [38].

In static pure aging, search points exceeding a predefined maximum lifespan (maximum number of iterations) τ are removed from the population. In stochastic aging, each search point x survives aging at the end of the iteration with a probability p_{die}. In order to keep the size of the population constant at a certain size μ, often new random search points with age 0 are introduced, if necessary.

The publications reviewed in the following subsections consider different kinds of aging in a minimal algorithmic framework by extending the $(\mu + 1)$ framework introduced in Algorithm 2. The extended framework in Algorithm 8 again uses a population of size μ. It works in rounds, where in each round all search points grow older, one new search point is generated as a random variation of existing search points, its age is decided, search points that are too old are removed, and new randomly generated search points are introduced to keep the number of search points constant at μ.

Algorithm 8: (μ+1) framework with aging

Parameters : Population size μ

Choose $x_1, \ldots, x_\mu \in \{0,1\}^n$ independently, uniformly at random, and let $P := \{x_1, \ldots, x_\mu\}$.
for *all* $x \in P$ **do**
    Set x.age $= 0$.
repeat
    for *all* $x \in P$ **do**
        set x.age $= x$.age $+ 1$. `/* Growing older */`
    Choose $x \in P$ uniformly at random.
    Create offspring $y :=$ mutate(x). `/* Variation */`
    Decide about the age of y. `/* (see details) */`
    Remove search points due to age. `/* (see details) */`
    if $|P| > \mu$ **then**
        Remove one $z \in P$ with minimum fitness. `/* Removal */`
    else
        Keep all search points in P.
        Fill up P with random points until $|P| = \mu$. `/* Birth */`
until *some termination criterion is met*

3.1 Static Pure Aging

In static pure aging, offspring inherit by default the age of their parent and are only assigned age 0 if their function value is strictly larger than that of their parents (see Algorithm 9). At the end of an iteration, search points that exceed the maximum age τ are removed deterministically (see Algorithm 10).

Algorithm 9: Static pure aging: age of offspring

Input: Parent x; offspring y

if $f(y) \geq f(x)$ **then**
    Set y.age $:= 0$.
else
    Set y.age $:= x$.age.

Algorithm 10: Static pure aging: removal due to age

Input: Population P; maximum lifespan τ

for *all* $x \in P$ **do**
    if $x.age > \tau$ **then**
        Set $P := P \setminus \{x\}$.

Horoba et al. [30] were the first to present a rigorous runtime analysis of static pure aging in artificial immune systems and consider its most important parameter: the maximum lifespan τ. They showed that the smaller the maximum age, the more the search process resembles pure random search and, thus, becomes ineffective. To be more precise, they demonstrated that τ needs to be large enough to allow the algorithm to create a better offspring. Considering a RIDGE-like function that includes a number of gaps of size k, they made their arguments more precise. They proved that, for such a function, τ needs to be sufficiently large in order for the algorithm to be successful, by presenting a proof that considers a typical run of a simple algorithm using static pure aging (see Algorithm 8) and standard bit mutations. Additionally, a common lower bound of $\tau = \omega(\mu n \log \mu)$ was derived – for smaller τ, the algorithm is unable to perform hill-climbing.

However, a maximum age that is too large severely limits the influence of the operator, as fewer search points are subject to removal by age. Moreover, there exist situations in which a small τ can prevent the algorithm under consideration from getting trapped in parts of the search space that keep it away from the global optimum. Again, this argument was made more precise by constructing an example function and proving that it can only be optimized efficiently using static pure aging if τ is sufficiently small.

Finally, it was shown that aging can be very sensitive to the maximum age of a search point and that it may be difficult to set this appropriately, i. e., the appropriate age can be within a very narrow range. Horoba et al. [30] demonstrated this by carefully devising a new example function by combining the two previous functions.

Building upon this work, Jansen and Zarges [33, 38] compared static pure aging with aging in evolutionary algorithms where new search points are always assigned age 0. It was shown that new random search points that are introduced during the birth phase typically have very low fitness and thus die out quickly. Using well-known example functions, they demonstrated that static pure aging is able to escape from local optima by recognizing stagnation and performing a kind of restart; however, when there are plateaus of constant function value, it mistakes the absence of progress in function values for stagnation and thus is not able to perform a random walk on the plateau. The performance of evolutionary aging is exactly opposite, i. e., it is not able to escape local optima but it can perform a random walk on a plateau. Based on these insights, a modified aging operator was introduced that provably shares the advantages of both aging mechanisms (see Algorithm 11): while the function values do not increase in both situations, being stuck in a local optimum additionally means that no new search points are created.

Algorithm 11: Genotypic aging: age of offspring

Input: Parent x; offspring y

if $f(y) \geq f(x)$ *and* $y \neq x$ **then**
  Set y.age $:= 0$.
else
  Set y.age $:= x$.age.

Later, Jansen and Zarges [35, 36, 39] analyzed the interplay of static pure aging with the replacement strategy used. It was demonstrated that static pure aging can achieve performance improvements that go beyond what restarts can accomplish [35]. Since it is often stated in the literature that aging increases the diversity within the population of search points, this is an important step in understanding how and why aging can make an algorithm more efficient. In this context, crossover plays an important role, as the main effect shown is based on the recombination (k-point crossover) of a local optimum and a randomly generated search point [35]. However, given the original definition of static pure aging, it is unclear how the age of a new search point is set in the case of more than one parent. Different strategies, including setting the age to the age of the older parent and setting it to the age of the better/worse parent, were introduced and analyzed in [36], where it was pointed out that even subtle differences can have a huge impact on the performance of the algorithm.

In [39], Jansen and Zarges argued that static pure aging can be subdivided into an aging and a replacement strategy. While the aging strategy determines the age of a new search point, the replacement strategy decides how it is introduced into the population. Considering a number of different implementations for both strategies, their interplay was analyzed. It was shown that not only the maximum age but also diversity with respect to age plays a key role and can make a difference between efficient and inefficient optimization. Different strategies that use the age to remove one of the search points with the worst function value from the population were considered. To compare these aging and replacement strategies, an example function was constructed and the performance for all possible combinations of operators was analyzed. As age diversity is mainly determined

by the interplay of the aging and the replacement strategies, a careful algorithm design and description was considered crucial to obtaining meaningful results.

Very recently, Corus et al. [11, 13] considered an artificial immune system using standard bit mutations and static pure aging in the context of the NP-hard number partitioning problem. Similarly to their analysis for hypermutations with mutation potential (see Section 2.3.2) they proved that their artificial immune system is also able to efficiently solve the same "hard" problem instances and constitutes a randomised polynomial time approximation scheme for the partitioning problem (for $\varepsilon \geq 4/n$).

3.2 Stochastic Aging

Stochastic aging usually uses the same mechanism as static pure aging to decide the age of an offspring (see Algorithm 9); however, search points are removed based on some probability p_{die} (see Algorithm 12).

Algorithm 12: Stochastic aging: removal due to age

Input: Population P; probability of dying p_{die}

1 **for** *all* $x \in P$ **do**

2 Set $P := P \setminus \{x\}$ with probability p_{die}.

Oliveto and Sudholt [52] presented the first theoretical analysis of stochastic aging. They showed that, just like static pure aging, stochastic aging can implicitly perform restarts but, more importantly, they also considered the question of what aging can achieve beyond performing standard restarts. They presented a framework for the analysis of stochastic aging using a given probability p_{die} and showed that stochastic aging can be effective in a natural setting (i. e., without crossover, which is not usually found in artificial immune systems) where restarts do not work.

Using the same classical example function that Jansen and Zarges [38] used for static pure aging, they provided guidance for parameterization and showed that stochastic aging as in Algorithm 12 is effective only for not too large population sizes, as the probability of performing a restart (i. e., the probability that all search points die roughly at the same time) is exponential in the population size μ. To tackle this problem, a hybrid aging operator (see Algorithm 13) combining ideas from static pure aging and stochastic aging was introduced. Like static pure aging, it protects a search point from dying for τ generations, but search points with an age larger than τ are removed from the population with probability p_{die}. The efficiency of this novel operator was demonstrated for example functions from the literature in both dynamic and static environments. These results hold for arbitrary population sizes. For the dynamic Balance function, it was shown that hybrid pure aging enables the algorithm to escape from a local optimum if all but one search point of the population die and, in the same iteration, the surviving search point moves out of the local optimum – something that an evolutionary algorithm using standard bit mutations is unable to achieve. Moreover, it was shown that static pure aging is inefficient in the dynamic setting considered. As a by-product of their analysis, Oliveto and Sudholt also remarked that the parameter setting for hybrid pure aging is at the opposite side of the spectrum from that for stochastic aging: while stochastic aging requires a high probability of surviving (close to 1), hybrid pure aging requires a low survival probability ($\Theta(1/\mu)$).

Algorithm 13: Hybrid pure aging: removal due to age

Input: Population P; lifespan τ; probability of dying p_{die}

1 **for** *all* $x \in P$ **do**

2 **if** $x.age > \tau$ **then**

3 Set $P := P \setminus \{x\}$ with probability p_{die}

More recently, Corus et al. [10, 15] extended the analysis of Oliveto and Sudholt [52] by considering the more general example function $\textsc{Cliff}_d$. They demonstrated that hybrid aging can be very efficient when coupled with local as well as standard bit mutations. For local mutations, they proved an expected optimization time of $O(n \log n)$ if the gap has linear size, i. e., when the function is most difficult for evolutionary algorithms using standard bit mutations. It is noted, that this asymptotically matches the lower bound for all unbiased mutation-based randomised search heuristics to optimise any function with a unique optimum [48]. For standard bit mutations, Corus et al. [10, 15] proved an expected optimisation time of $O(n^{1+\varepsilon})$, $\varepsilon > 0$ constant, if the gap has linear size. The study was further expanded in [15] by adding a genotype diversity mechanism (as proposed in the original Opt-IA). Here, it was shown that the algorithm can still escape from the local optima provided that the population size is not too large.

4 Theoretical Analyses of Complete AIS

While most of the theoretical work so far has considered only specific ingredients of artificial immune systems from the literature, some work has analyzed complete AIS as used in applications. Early examples examined the B-cell algorithm [47]. More recently, a study of Opt-IA [18] was presented. These publications shed a more detailed light on the strengths and weaknesses of immune-inspired approaches compared with other randomized search heuristics such as evolutionary algorithms or random local search by examining the interplay between different mechanisms. The following sections present pseudocode for both algorithms considered in this way, and an overview of the results obtained.

4.1 The B-Cell Algorithm

The B-cell algorithm (BCA; see Algorithm 14) uses a population of size μ, generates λ clones for each member of the population, and applies standard bit mutations to one random clone of each member and somatic contiguous hypermutations to all clones. It applies plus-selection between each member of the population and its clones. Jansen et al. [31] proposed a variant of the BCA where contiguous hypermutations are only applied with some constant probability $0 < p < 1$ to the search point undergoing standard bit mutations. Thus, in this version, one of the offspring is subject to standard bit mutations with only probability $1 - p$. We call this variant BCA*.

Algorithm 14: The B-cell algorithm (BCA)

Parameters : Population size μ; offspring population size λ;
mutation probability $r \in (0, 1]$

Choose $x_1, \ldots, x_\mu \in \{0,1\}^n$ independently uniformly at random (u.a.r.)
repeat
  for $i \in \{1, \ldots, \mu\}$ **do** `/* Clonal expansion */`
    for $j \in \{1, \ldots, \lambda\}$ **do**
      Set $y_{i,j} := x_i$.
  for $i \in \{1, \ldots, \mu\}$ **do** `/* Standard bit mutations */`
    Select $j \in \{1, \ldots, \lambda\}$ u.a.r.
    Perform SBM($y_{i,j}$).
  for $i \in \{1, 2, \ldots, \mu\}$ **do** `/* Contiguous hypermutations */`
    for $j \in \{1, 2, \ldots, \lambda\}$ **do**
      Perform CHM($y_{i,j}$). `// see Algorithm 5`
  for $i \in \{1, 2, \ldots, \mu\}$ **do** `/* Selection */`
    if $f(x_i) \leq \max\{f(y_{i,1}), \ldots, f(y_{i,\lambda})\}$ **then**
      Set $x_i := y_{i,j}$, where $f(y_{i,j}) = \max\{f(y_{i,1}), \ldots, f(y_{i,\lambda})\}$,
      break ties u.a.r.
until *some termination criterion is met*

4.1.1 Vertex Cover

The work of Jansen et al. [31] constitutes the first runtime analysis of a clonal selection algorithm from the literature without any simplifications. It considers the performance of the BCA and BCA* on the vertex cover problem and compares it with known results for evolutionary algorithms.

In the vertex cover problem, we are given an undirected graph $G = (V, E)$ with a set V of $n = |V|$ vertices and a set E of $m = |E|$ edges. A cover is a subset of nodes, $V' \subseteq V$, such that each edge $e \in E$ is covered by at least one node in V', i. e., $e \cap V' \neq \emptyset$. One is interested in finding a small cover, i. e., a cover $V^* \subseteq V$ such that no smaller subset of V can be a cover, i. e., $\forall V' \subseteq V\colon (|V'| < |V^*|) \Rightarrow (\exists e \in E\colon e \cap V^* = \emptyset)$.

In their work, Jansen et al. [31] used the standard node-based representation for vertex cover that assumes that each $x \in \{0,1\}^n$ encodes the node selection $V(x) = \{v_i \in V \mid x[i] = 1\}$. For a bit string x that encodes a cover $V(x)$, we use its size $|V(x)|$ as its fitness; otherwise, the number of edges that are not covered is used as a penalty term, yielding the following fitness function that is to be minimized:

$$f(x) = \begin{cases} |V(x)| & \text{if } \forall e \in E\colon V(x) \cap e \neq \emptyset, \\ (|V| + 1) \cdot |\{e \in E \mid V(x) \cap e = \emptyset\}| & \text{otherwise.} \end{cases}$$

Since, for the BCA, it is easy to flip contiguous bits but difficult to simultaneously flip a few bits which are far apart, the mapping between a bit in x and a node in V can have a significant influence on the performance of the algorithm. Thus, Jansen et al. [31] suggested an ordering heuristic to determine a suitable mapping instead of making an arbitrary choice. The main idea behind this heuristic is that nodes that are close to each other and share many neighbors are likely to be ordered together (see [31] for a formal definition and illustrative examples).

Using this encoding, Jansen et al. [31] considered a sequence of increasingly complex instances of the vertex cover problem that have been studied as example instances for different kinds of randomized search heuristics to explore their limits. The simplest example (introduced by Friedrich et al. [26]) is a bipartite graph where a small set of nodes

V_1 (with size $|V_1| = \varepsilon n$) is completely connected to a larger set of nodes V_2 (with size $|V_2| = (1-\varepsilon)n$). The number of nodes n and the parameter ε that defines the imbalance in the sizes of the two sets are parameters. Friedrich et al. [26] proved that the $(1+1)$ EA is easily caught in the local optimum and therefore is very inefficient with respect to expected optimization time. The same holds for random local search. Both heuristics require the introduction of restarts when stuck in a local optimum to become efficient on this problem instance [49]. The BCA, with any polynomial population size μ and any not too large number of clones ($\lambda = O(1)$), has expected optimization time $O(\mu n^2 \log n)$ and does not require restarts.

To make the difference between the $(1+1)$ EA and the BCA more pronounced, one can "amplify" the result by considering a number of copies of the bipartite graph and adding a small number of additional edges to make the graph connected. For this graph, the $(1+1)$ EA has an exponential expected optimization time even if it is equipped with an optimal restart strategy [50]. For a number l of copies of the bipartite graph, where each copy has h nodes (so that the total graph has $n = h \cdot l$ nodes), the BCA has expected optimization time $O(\mu n^2 (l + \log(n)))$, again for any polynomial population size μ and any not too large number of clones ($\lambda = O(1)$).

The $(1+1)$ EA is not a very typical evolutionary algorithm, since it employs neither a proper population of solutions nor crossover. For evolutionary algorithms making use of both, more complex vertex cover instances become solvable. Oliveto et al. [49] proved that an evolutionary algorithm with a population size of μ that applies crossover with a small probability p_c is able to find an optimal solution for a more complex vertex cover instance in polynomial time with very high probability, namely in time $O(\mu^2 n/p_c)$, where the population size is at least $\mu \geq n^{1+\varepsilon}$ and the probability of applying crossover is at most $p_c \leq 1/(\mu\sqrt{n}\log n)$. Note that the upper bound is $\omega(n^{4.5}\log n)$, which is far from being efficient from a practical point of view. The $(1+1)$ EA is provably very inefficient on this problem instance. The BCA, on the other hand, finds an optimum for this instance in expected time $O(\mu n^3)$, which can be as small as $O(n^3)$ if the population size μ is small ($\mu = O(1)$).

As seen in Section 2.2, contiguous hypermutations are a rather inefficient hill-climber, and thus so is the "pure" BCA. Using BCA* instead of the BCA reduces the upper bound to $O(\mu n^2 \log n)$, which becomes $O(n^2 \log n)$ for small population sizes ($\mu = O(1)$). This modification improves the hill-climbing abilities of the BCA considerably without compromising its search capabilities in a significant way. We remark that for $\mu = O(1)$ and $\lambda = O(1)$, BCA* also improves the expected optimization times for OneMax and LeadingOnes to $\Theta(n \log n)$ and $\Theta(n^2)$, respectively.

4.1.2 Longest Common Subsequence

A comparison similar to the one presented by Jansen et al. [31] for the vertex cover problem has been performed for the longest common subsequence problem [41]. Here, Jansen and Zarges showed that the BCA outperforms a large class of evolutionary algorithms using mutation and crossover on previously introduced hard problem instances.

In the longest common subsequence problem, we are given a set of m sequences of potentially different lengths over a common finite alphabet Σ, i. e., $X_1, X_2, \ldots, X_m \subseteq \Sigma^*$. By $|Y|$ we denote the length of a sequence Y, i. e., $|Y| = l$ for $Y = y[1]y[2]\cdots y[l] \in \Sigma^l$. A sequence $Y = y[1]y[2]\ldots y[l] \in \Sigma^l$ is called a subsequence of a sequence $X = x[1]x[2]\cdots x[n] \in \Sigma^n$ if there are indices $0 < i_1 < i_2 < \cdots < i_l \leq n$ such that $y[j] = x[i_j]$ holds for all $j \in \{1, 2, \ldots, l\}$. The sequence of indices proving that Y is a subsequence of X need not be unique. A sequence Y is a common subsequence of $X_1, X_2, \ldots, X_m$ if it is a subsequence of X_i for all $i \in \{1, 2, \ldots, m\}$. It is a longest common subsequence if all common subsequences of $X_1, X_2, \ldots, X_m$ do not have greater length.

Jansen and Zarges [41] used $S = \{0,1\}^n$, where n is the length of a shortest sequence in the input, as the search space. Let $X_1 = x[1]x[2]\cdots x[n] \in \Sigma^n$ denote the letters in the

sequence X_1. For a search point $s = s[1]s[2]\cdots s[n] \in \{0,1\}^n$, let $I_1 = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$ (with $i_1 < i_2 < \cdots < i_l$) denote the positions of 1-bits in s, i. e., $s[i] = 1$ for all $i \in I_1$ and $s[i] = 0$ for all $i \in \{1, 2, \ldots, n\} \setminus I_1$. The search point s encodes the sequence $x[i_1]x[i_2]\cdots x[i_l]$, a subsequence of X_1. Let $c(s)$ denote the sequence encoded by s. If $c(s)$ is a subsequence of all $X_1, X_2, \ldots, X_m$, it encodes a feasible solution, otherwise $c(s)$ is infeasible. The all-zero bit string encodes a trivial empty solution.

We discuss only one of the three fitness functions considered in [41], as the other two are either very complicated or merely of theoretical interest. The function f_{MAX} determines the maximum length k of a prefix of $c(s)$ such that $c(s)_{(k)}$ is a common subsequence of $X_1, X_2, \ldots, X_m$. This length minus the length of the remaining suffix of $c(s)$ is the function value:

$$\begin{aligned}
&\text{MAX}(c(s), X_1, X_2, \ldots, X_m) \\
&\qquad = \min\{\max\{k \mid c(s)_{(k)} \text{ is subsequence of } X_i\} \mid i \in \{1, \ldots, m\}\}, \\
&f_{\text{MAX}}(s) \\
&\qquad = \text{MAX}(c(s), X_1, X_2, \ldots, X_m) - (|c(s)| - \text{MAX}(c(s), X_1, X_2, \ldots, X_m)).
\end{aligned}$$

Jansen and Zarges [41] considered four hard instances from the literature [32], two for the theoretically motivated fitness function omitted here and two for the other two (including the one defined above):

- E_{MAX}:
$$X_1 = 0^{(8/32)n}1^{(24/32)n} \text{ and } X_2 = 1^{(24/32)n}0^{(5/32)n}1^{(13/32)n},$$
where n is a multiple of 32;
- A_{MAX}:
$$X_1 = 0^{(1/l)n}1^{((l-1)/l)n} \text{ and } X_2 = 1^{((l-1)/l)n}0^{(5/(8l))n}1^{((4l-3)/(8l))n},$$
where $l := \lceil (3/\varepsilon) - (1/2) \rceil$ for some $\varepsilon > 0$ constant and n a multiple of $8l$.

It is known that a large class of evolutionary algorithms fails to locate an optimal solution of E_{MAX} efficiently; for A_{MAX}, this class even fails to approximate an optimal solution up to a factor of $2 - \varepsilon$ for any constant $\varepsilon > 0$ [32].

Jansen and Zarges [41] proved that the BCA is not efficient if random initialization of the population is used; for A_{MAX}, it also fails to find a good approximation – just like evolutionary algorithms. However, they showed that the BCA is very efficient if started with trivial empty candidate solutions. For both E_{MAX} and A_{MAX}, the expected optimization time of the BCA is $O(\mu\lambda n^2 \log n)$ for all settings of $\mu = n^{O(1)}$, $\lambda = n^{O(1)}$ with $\mu\lambda = \omega(n \log n)$. The algorithm benefits from deterministic initialization because contiguous hypermutations are able to introduce a linear number of 1-bits into a region where they are needed in a single step. As a by-product of their analyses, Jansen and Zarges [41] noted that the concrete choices of μ and λ make no difference as long as $\mu \cdot \lambda$ remains unchanged – in evolutionary computation, these choices usually have a very different effect.

While empirical observations for the longest common subsequence problem indicate that evolutionary algorithms perform better if started with trivial empty candidate solutions, this is not the case for the instances considered, and deterministic initialization does not lead to an improved behavior.

4.1.3 Dynamic Optimization

The BCA and its variant BCA* have also been considered in the context of dynamic optimization [42, 45]. Here, Jansen and Zarges particularly discussed why fixed-budget analysis is more appropriate for dynamic environments, where the limited time budget refers to the generations directly after a change in the fitness landscape. They introduced

a novel dynamic bistable example function that exhibits phases of stability and rapid change. Motivated by earlier results, they investigated whether artificial immune systems have an advantage in situations of rapid change. A large number of concrete theoretical results for different combinations of execution platforms and parameters of the fitness function were presented. The specific way the optimum moves in the nonstable phases tends to be helpful for contiguous hypermutations, but within the analytical framework no clear advantage could be observed. The concrete contributions of [42, 45] are discussed in more detail in Section 5.5 of this book.

4.2 Opt-IA

The name Opt-IA [18] encompasses several clonal selection algorithms following similar ideas and using roughly the same operators. Just like the B-cell algorithm, Opt-IA is mostly used in the context of optimization and uses a bit string representation. We give a description of the algorithm's bare bones in Algorithm 15. For each search point in the population, a large number of clones are created that are then subject to mutation. Usually a static cloning operator is used, i. e., the number of clones is independent of the fitness; however, some versions of Opt-IA employ some form of fitness-dependent cloning, where a search point is selected for cloning with a probability that is proportional to its fitness (see [3] for an overview). Depending on the specific variant used, Opt-IA uses two different types of mutation operator: hypermutations with mutation potential and a form of contiguous hypermutations (often called hypermacromutation). Usually, both mutation operators are applied independently and separately to the clones. Opt-IA additionally introduces the concept of aging to clonal selection algorithms, as discussed in Section 3. Aging operators aim at increasing the diversity within the population by removing "too old" search points. If aging results in too few search points in the population, the population is filled up with new random search points. Moreover, usually no duplicates are allowed in the population.

Algorithm 15: Opt-IA

Parameters : Population size μ; offspring population size λ; mutation flags H, M

Choose $P = \{x_1, \ldots, x_\mu\}$ independently, uniformly at random (u.a.r.).
repeat
  for $i \in \{1, \ldots, \mu\}$ **do** `/* Clonal selection and expansion */`
    Generate λ clones of x_i.
    Place the clones in a clonal pool $C_i = \{y_{i,1}, \ldots, y_{i,\lambda}\}$.
    $C_i^H = \emptyset$. $C_i^M = \emptyset$.
    for $j \in \{1, \ldots, \lambda\}$ **do** `/* Affinity maturation */`
      if H **then**
        $\hat{y}_{i,j} \leftarrow$ Apply hypermutations with mutation potential to $y_{i,j}$.
        Add $\hat{y}_{i,j}$ to C_i^H.
      if M **then**
        $\tilde{y}_{i,j} \leftarrow$ Apply contiguous hypermutations to $y_{i,j}$.
        Add $\tilde{y}_{i,j}$ to C_i^M.
  Apply aging to P, C_i^H, and C_i^M. `/* Metadynamics */`
  Set $P = P \cup C_1^H \cup \ldots \cup C_\mu^H \cup C_1^M \cup \ldots \cup C_\mu^M$. `/* Selection */`
  if $|P| > \mu$ **then**
    Keep the μ best search points from P, breaking ties u.a.r. and removing duplicates.
  else
    Keep all search points in P.
    Fill up P with random points until $|P| = \mu$.
until *some termination criterion is met*

Corus et al. [10, 15] considered the first runtime analysis of Opt-IA using only hypermutations with a static mutation potential (where the number of bits flipped is linear in the problem size, $c \cdot n$ for constant $c > 0$), including the FCM mechanism and ensuring that only distinct bits are flipped (TABU variant). Moreover, they replaced the standard static pure aging operator by hybrid aging, as introduced by Oliveto and Sudholt [52] (see Algorithm 13).

After considering standard example functions such as OneMax, LeadingOnes, Cliff_d, and Jump_k for this variant of Opt-IA (with and without genotype diversity), the study highlights problems where the use of the complete Opt-IA variant is crucial. For a carefully constructed novel example function called HiddenPath, it was shown that Opt-IA with appropriate parameterization has an expected polynomial optimization time, while the algorithm missing either aging or hypermutations requires at least superpolynomial time. To give a complete picture, the extension in [15] introduced another class of functions (called HyperTrap_y), for which, with overwhelming probability, Opt-IA is inefficient, while the simple (1+1) EA using standard bit mutations is efficient.

Corus et al. [10, 15] also considered a simple trap function, as such a function was used when Opt-IA was originally introduced. They proved an expected optimization time of $O(\mu n^2 \log n)$ for $\tau = \Omega(n^2)$, $c = 1$, and $\lambda = 1$ and pointed out that this does not match the empirical results reported in [16], where Opt-IA was unable to optimize trap functions for $n > 50$. It was conjectured that this was due either to not using FCM or too small a time budget.

In [12], Corus et al. analyzed Opt-IA using "fast" hypermutations with mutation potential (see Section 2.3.3) on previously considered example functions such as HiddenPath and Cliff_d. The authors particularly pointed out that, in order to effectively work with aging on Cliff_d, it was crucial not to use hypermutations with FCM as the FCM

mechanism does not allow worsening of the fitness value[1]. Thus, the operator performed all n mutation steps and returned the best sampled search point instead of the first improved one. Later, Corus et al. [14] demonstrated that their inversely fitness-proportional mutation potential (see Section 2.3.1) together with aging was able to optimize $\textsc{Cliff}_d$ with $d = \Theta(n)$ in expected polynomial time even if FCM is used.

5 Summary

In this chapter, we have provided an overview of the state of the art in the theory of immune-inspired randomized search heuristics for optimization. In this context, most algorithms are inspired by the so-called clonal selection principle, which describes the basic features of the adaptive immune response. A large variety of different clonal selection algorithms have been introduced, and over the last decade some significant progress has been made on the theoretical foundations of such algorithms. Initially, most theoretical studies concentrated on two defining aspects of artificial immune systems: hypermutation operators (inversely fitness-proportional mutations, contiguous hypermutations, and hypermutations with mutation potential) and a diversity mechanism called aging (static pure aging and stochastic aging). More recently, insights into the interplay between different operators have allowed the first analyses of "complete" artificial immune systems as published in the literature – this particularly includes analyses of the B-cell algorithm and Opt-IA.

Theoretical analyses have contributed to significant insights into the working principles of immune-inspired operators and algorithms. For example, a common observation in the literature is that typical immune-inspired operators such as hypermutations and aging allow us to efficiently escape from local optima – particularly when compared to evolutionary algorithms – but may have difficulties during the exploitation phase. The introduction of fixed-budget analysis (discussed in more detail in Chapter 5 of this book) has particularly contributed to our understanding of their strengths and weaknesses. In many cases, these insights have contributed to the development of improved versions of the operators or hybrid variants that combine immune-inspired mechanisms with techniques used in evolutionary computation and other randomized search heuristics. However, more research into the strengths and weaknesses of immune-inspired algorithms is needed, particularly in the context of combinatorial optimization. It would be interesting to see on what kind of problems these algorithms excel over other nature-inspired randomized search heuristics such as evolutionary algorithms. It is also often argued that immune-inspired algorithms are especially suited for multimodal or dynamic optimization problems. Further investigations in these directions are promising directions for future research.

References

[1] H. S. Bernardino and H. J. C. Barbosa. Artificial immune systems for optimization. In R. Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193 of *Studies in Computational Intelligence*, pages 389–411. Springer, 2009.

[2] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, volume 6238 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2010.

[3] J. Brownlee. Clonal selection algorithms. Technical Report 070209A, Swinburne University of Technology, Victoria, Australia, 2007.

[1] This observation holds for both "classical" and "fast" hypermutations with mutation potential.

[4] F. M. Burnet. *The Clonal Selection Theory of Acquired Immunity*. Cambridge University Press, 1959.

[5] E. B. Clark. *A Framework for Modelling Stochastic Optimisation Algorithms With Markov Chains*. PhD thesis, University of York, 2008.

[6] E. B. Clark, A. Hone, and J. Timmis. A Markov chain model of the B-cell algorithm. In *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS 2005)*, volume 3627 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2005.

[7] C. A. C. Coello and N. C. Cortés. Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6(2):163–190, 2005.

[8] D. Corus, J. He, T. Jansen, P. S. Oliveto, D. Sudholt, and C. Zarges. On easiest functions for somatic contiguous hypermutations and standard bit mutations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015)*, pages 1399–1406. ACM, 2015.

[9] D. Corus, J. He, T. Jansen, P. S. Oliveto, D. Sudholt, and C. Zarges. On easiest functions for mutation operators in bio-inspired optimisation. *Algorithmica*, 78(2):714–740, 2017.

[10] D. Corus, P. S. Oliveto, and D. Yazdani. On the runtime analysis of the opt-IA artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017)*, pages 83–90. ACM, 2017.

[11] D. Corus, P. S. Oliveto, and D. Yazdani. Artificial immune systems can find arbitrarily good approximations for the NP-hard partition problem. In *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN 2018), Part II*, volume 11102 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2018.

[12] D. Corus, P. S. Oliveto, and D. Yazdani. Fast artificial immune systems. In *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN 2018), Part II*, volume 11102 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2018.

[13] D. Corus, P. S. Oliveto, and D. Yazdani. Artificial immune systems can find arbitrarily good approximations for the NP-hard number partitioning problem. *Artificial Intelligence*, 274:180–196, 2019.

[14] D. Corus, P. S. Oliveto, and D. Yazdani. On inversely proportional hypermutations with mutation potential. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2019)*, pages 215–223. ACM, 2019.

[15] D. Corus, P. S. Oliveto, and D. Yazdani. When hypermutations and ageing enable artificial immune systems to outperform evolutionary algorithms. *Theoretical Computer Science*, 2019. In press. `https://doi.org/10.1016/j.tcs.2019.03.002`.

[16] V. Cutello, G. Nicosia, and M. Pavone. Exploring the capability of immune algorithms: A characterization of hypermutation operators. In *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS 2004)*, volume 3239 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2004.

[17] V. Cutello, G. Nicosia, M. Romeo, and P. S. Oliveto. On the convergence of immune algorithms. In *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)*, pages 409–415. IEEE, 2007.

[18] V. Cutello, M. Pavone, and J. Timmis. An immune algorithm for protein structure prediction on lattice models. *IEEE Transactions on Evolutionary Computation*, 11(1):101–117, 2007.

[19] D. Dasgupta, editor. *Artificial Immune Systems and Their Applications*. Springer, 1998.

[20] D. Dasgupta and L. F. Niño. *Immunological Computation: Theory and Applications*. Auerbach, 2008.

[21] L. N. de Castro and J. Timmis. An artificial immune network for multimodal function optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC 2002)*, pages 699–704. IEEE Press, 2002.

[22] L. N. de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, 2002.

[23] L. N. de Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002.

[24] M. Elberfeld and J. Textor. Negative selection algorithms on strings with efficient training and linear-time classification. *Theoretical Computer Science*, 412(6):534–542, 2011.

[25] D. A. B. Fernandes, M. M. Freire, P. A. P. Fazendeiro, and P. R. M. Inácio. Applications of artificial immune systems to computer security: A survey. *Journal of Information Security and Applications*, 35:138–159, 2017.

[26] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18(4):617–633, 2010.

[27] F. Gu, J. Greensmith, and U. Aickelin. Theoretical formulation and analysis of the deterministic dendritic cell algorithm. *Biosystems*, 111(2):127–135, 2013.

[28] J. He, T. Chen, and X. Yao. On the easiest and hardest fitness functions. *IEEE Transactions on Evolutionary Computation*, 19(2):295–305, 2015.

[29] L. Hong and J. Kamruzzaman. Convergence of elitist clonal selection algorithm based on martingale theory. *Engineering Letters*, 21(4):181–184, 2013.

[30] C. Horoba, T. Jansen, and C. Zarges. Maximal age in randomized search heuristics with aging. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, pages 803–810. ACM, 2009.

[31] T. Jansen, P. S. Oliveto, and C. Zarges. On the analysis of the immune-inspired B-cell algorithm for the vertex cover problem. In *Proceedings of the 10th International Conference on Artificial Immune Systems (ICARIS 2011)*, volume 6825 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 2011.

[32] T. Jansen and D. Weyland. Analysis of evolutionary algorithms for the longest common subsequence problem. *Algorithmica*, 57:170–186, 2010.

[33] T. Jansen and C. Zarges. Comparing different aging operators. In *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, volume 5666 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2009.

[34] T. Jansen and C. Zarges. A theoretical analysis of immune inspired somatic contiguous hypermutations for function optimization. In *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, volume 5666 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2009.

[35] T. Jansen and C. Zarges. Aging beyond restarts. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, pages 705–712. ACM, 2010.

[36] T. Jansen and C. Zarges. On the benefits of aging and the importance of details. In *Proceedings of the 9th International Conference on Artificial Immune Systems (ICARIS 2010)*, volume 6209 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 2010.

[37] T. Jansen and C. Zarges. Analyzing different variants of immune inspired somatic contiguous hypermutations. *Theoretical Computer Science*, 412(6):517–533, 2011.

[38] T. Jansen and C. Zarges. On benefits and drawbacks of aging strategies for randomized search heuristics. *Theoretical Computer Science*, 412(6):543–559, 2011.

[39] T. Jansen and C. Zarges. On the role of age diversity for effective aging operators. *Evolutionary Intelligence*, 4(2):99–125, 2011.

[40] T. Jansen and C. Zarges. Variation in artificial immune systems: Hypermutations with mutation potential. In *Proceedings of the 10th International Conference on Artificial Immune Systems (ICARIS 2011)*, volume 6825 of *Lecture Notes in Computer Science*, pages 132–145. Springer, 2011.

[41] T. Jansen and C. Zarges. Computing longest common subsequences with the B-cell algorithm. In *Proceedings of the 11th International Conference on Artificial Immune Systems (ICARIS 2012)*, volume 7597 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2012.

[42] T. Jansen and C. Zarges. Evolutionary algorithms and artificial immune systems on a bi-stable dynamic optimisation problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, pages 975–982. ACM, 2014.

[43] T. Jansen and C. Zarges. Reevaluating immune-inspired hypermutations using the fixed budget perspective. *IEEE Transactions on Evolutionary Computation*, 18(5):674–688, 2014.

[44] T. Jansen and C. Zarges. Understanding randomised search heuristics. lessons from the evolution of theory: A case study. In *Proceedings of the 20th International Conference on Soft Computing (MENDEL 2014)*, pages 293–298, 2014.

[45] T. Jansen and C. Zarges. Analysis of randomised search heuristics for dynamic optimisation. *Evolutionary Computation*, 23:513–541, 2015.

[46] N. Jerne. Towards a network theory of the immune system. *Annals of Immunology*, 125C(1–2):373–389, 1974.

[47] J. Kelsey and J. Timmis. Immune inspired somatic contiguous hypermutation for function optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, volume 2723 of *Lecture Notes in Computer Science*, pages 207–218. Springer, 2003.

[48] P. K. Lehre and C. Witt. Black-box search by unbiased variation. *Algorithmica*, 64(4):623–642, 2012.

[49] P. S. Oliveto, J. He, and X. Yao. Analysis of population-based evolutionary algorithms for the vertex cover problem. In *Proceedings of the Congress on Evolutionary Computation (CEC 2008)*, pages 1563–1570. IEEE Press, 2008.

[50] P. S. Oliveto, J. He, and X. Yao. Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems. *IEEE Transactions Evolutionary Computation*, 13(5):1006–1029, 2009.

[51] P. S. Oliveto, P. K. Lehre, and F. Neumann. Theoretical analysis of rank-based mutation – combining exploration and exploitation. In *Proceedings of the Congress on Evolutionary Computation (CEC 2009)*, pages 1455–1462. IEEE, 2009.

[52] P. S. Oliveto and D. Sudholt. On the runtime analysis of stochastic ageing mechanisms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, pages 113–120. ACM, 2014.

[53] G. C. Silva and D. Dasgupta. A survey of recent works in artificial immune systems. In P. P. Angelov, editor, *Handbook on Computational Intelligence*, chapter Chapter 15, pages 547–586. World Scientific, 2016.

[54] J. Textor. Efficient negative selection algorithms by sampling and approximate counting. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN 2012)*, volume 7491 of *Lecture Notes in Computer Science*, pages 32–41. Springer, 2012.

[55] J. Timmis, A. Hone, T. Stibor, and E. Clark. Theoretical advances in artificial immune systems. *Theoretical Computer Science*, 403(1):11–32, 2008.

[56] M. Villalobos-Arias, C. A. C. Coello, and O. Hernández-Lerma. Convergence analysis of a multiobjective artificial immune system algorithm. In *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS 2004)*, volume 3239 of *Lecture Notes in Computer Science*, pages 226–235. Springer, 2004.

[57] M. Villalobos-Arias, C. A. C. Coello, and O. Hernández-Lerma. Asymptotic convergence of some metaheuristics used for multiobjective optimization. In *Proceedings of the 8th International Workshop on Foundations of Genetic Algorithms (FOGA 2005)*, volume 3469 of *Lecture Notes in Computer Science*, pages 95–111. Springer, 2005.

[58] X. Xia and Y. Zhou. On the effectiveness of immune inspired mutation operators in some discrete optimization problems. *Information Sciences*, 426:87–100, 2018.

[59] C. Zarges. Rigorous runtime analysis of inversely fitness proportional mutation rates. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN 2008)*, volume 5199 of *Lecture Notes in Computer Science*, pages 112–122. Springer, 2008.

[60] C. Zarges. On the utility of the population size for inversely fitness proportional mutation rates. In *Proceedings of the 10th International Workshop on Foundations of Genetic Algorithms (FOGA 2009)*, pages 39–46. ACM Press, 2009.

[61] C. Zarges. *Theoretical Foundations of Artificial Immune Systems*. PhD thesis, TU Dortmund, Germany, 2011.